# DIGITAL SYSTEM DESIGN LABORATORY

# LAB 6

## SINGLE CYCLE MICROPROCESSOR DESIGN

# I. LAB OBJECTIVES

This Lab experiments are intended to design and test a Single Cycle Microprocessor

# II. DESCRIPTION

Single Cycle Microprocessor datapath to be implemented is in figure 2.1.

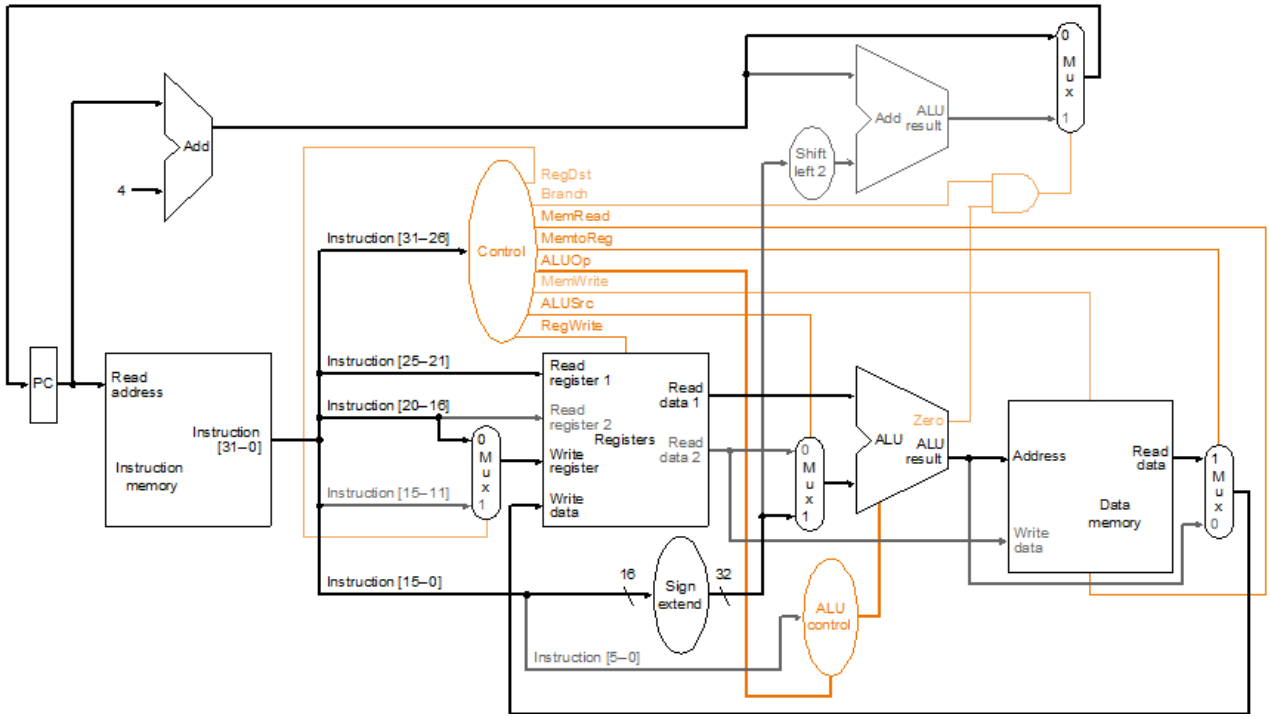



**Figure 2.1: Single Cycle Microprocessor DataPath**

# III. LAB PROCEDURE

## III.1 EXPERIMENT NO. 1

**III.1.1 AIM:** To understand and write the assembly code using MIPS Instruction set

**register number of MIPS compiler conventions**

| Name | Register number | Usage |
|---|---|---|
| $zero | 0 | the constant value 0 |
| $v0-$v1 | 2-3 | values for results and expression evaluation |
| $a0-$a3 | 4-7 | arguments |
| $t0-$t7 | 8-15 | temporaries |
| $s0-$s7 | 16-23 | saved (by callee) |
| $t8-$t9 | 24-25 | more temporaries |
| $gp | 28 | global pointer |
| $sp | 29 | stack pointer |
| $fp | 30 | frame pointer |
| $ra | 31 | return address |

## MIPS Assembly Language Sumarize:

| MIPS assembly language | | | | |
|---|---|---|---|---|
| Category | Instruction | Example | Meaning | Comments |
| | add | `add $s1, $s2, $s3` | `$s1 = $s2 + $s3` | Three operands; data in registers |
| Arithmetic | subtract | `sub $s1, $s2, $s3` | `$s1 = $s2 - $s3` | Three operands; data in registers |
| | add immediate | `addi $s1, $s2, 100` | `$s1 = $s2 + 100` | Used to add constants |
| | load word | `lw $s1, 100($s2)` | `$s1 = Memory[$s2 + 100]` | Word from memory to register |
| | store word | `sw $s1, 100($s2)` | `Memory[$s2 + 100] = $s1` | Word from register to memory |
| Data transfer | load byte | `lb $s1, 100($s2)` | `$s1 = Memory[$s2 + 100]` | Byte from memory to register |
| | store byte | `sb $s1, 100($s2)` | `Memory[$s2 + 100] = $s1` | Byte from register to memory |
| | load upper immediate | `lui $s1, 100` | $s1 = 100 * $2^{16}$ | Loads constant in upper 16 bits |
| | branch on equal | `beq $s1, $s2, 25` | if (`$s1 == $s2`) go to PC + 4 + 100 | Equal test; PC-relative branch |
| Conditional branch | branch on not equal | `bne $s1, $s2, 25` | if (`$s1 != $s2`) go to PC + 4 + 100 | Not equal test; PC-relative |
| | set on less than | `slt $s1, $s2, $s3` | if (`$s2 < $s3`) `$s1` = 1; else `$s1` = 0 | Compare less than; for beq, bne |
| | set less than immediate | `slti $s1, $s2, 100` | if (`$s2 < 100`) `$s1` = 1; else `$s1` = 0 | Compare less than constant |
| | jump | `j 2500` | go to 10000 | Jump to target address |
| Uncondi- | jump register | `jr $ra` | go to `$ra` | For switch, procedure return |
| tional jump | jump and link | `jal 2500` | `$ra` = PC + 4; go to 10000 | For procedure call |

Operation code (Op code) sumarize:

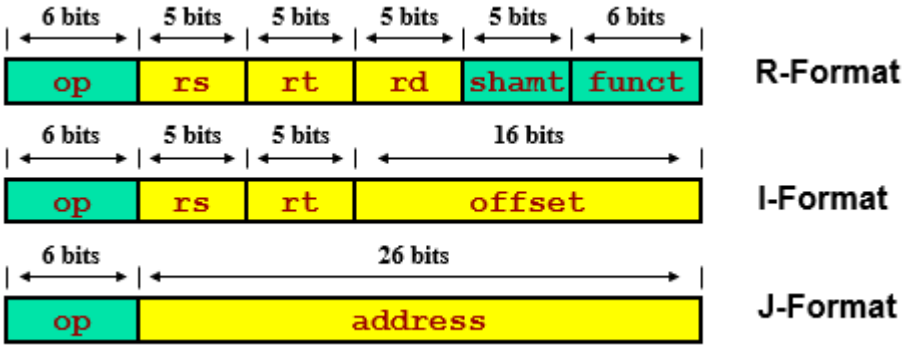
| Op | Opcode name | Value |
|---|---|---|
| 000000 | R-format | Rformat1 |
| 000010 | jmp | JUMP1 |
| 000100 | beq | BEQ1 |
| 100011 | lw | Mem1 |
| 101011 | sw | Mem1 |

## ALU opcode and Function

| $ALUOp_{1:0}$ | Meaning |
|---|---|
| 00 | Add |
| 01 | Subtract |
| 10 | Look at Funct |
| 11 | Not Used |

| $ALUOp_{1:0}$ | Funct | $ALUControl_{2:0}$ |
|---|---|---|
| 00 | X | 010 (Add) |
| X1 | X | 110 (Subtract) |
| 1X | 100000 (add) | 010 (Add) |
| 1X | 100010 (sub) | 110 (Subtract) |
| 1X | 100100 (and) | 000 (And) |
| 1X | 100101 (or) | 001 (Or) |
| 1X | 101010 (slt) | 111 (SLT) |

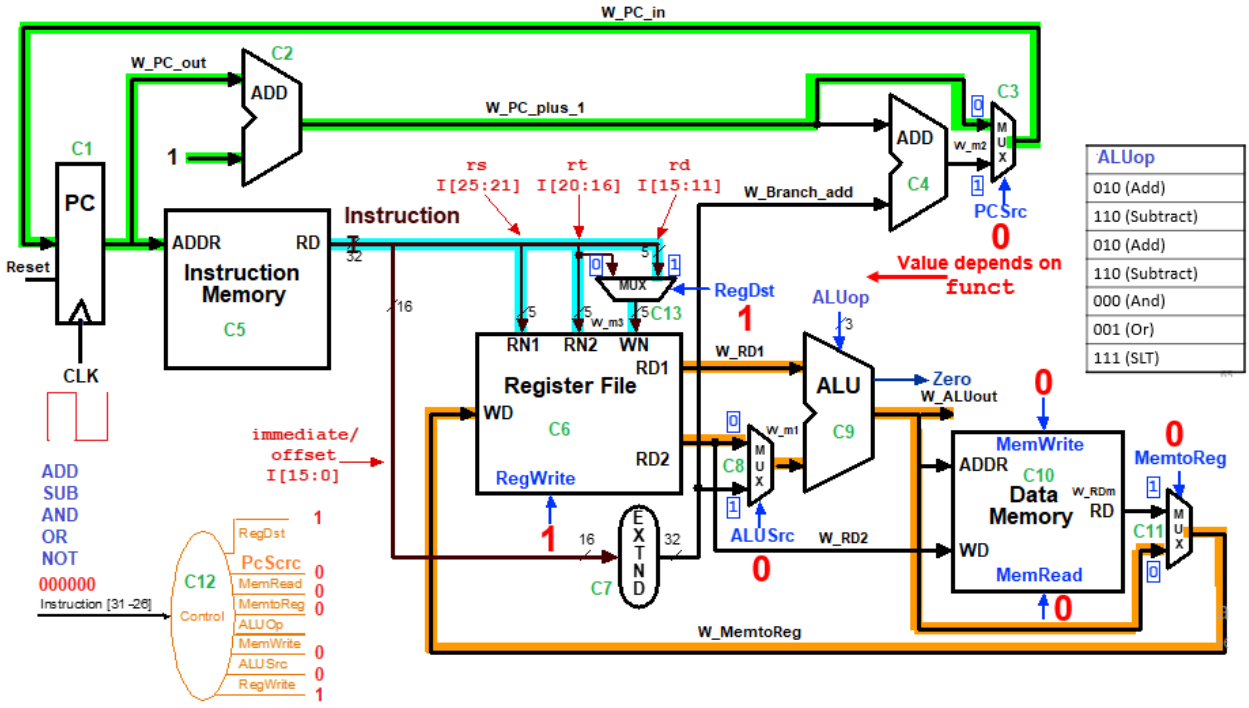
Instruction Formats



## III.1 EXPERIMENT NO. 1

### III.1.1 AIM: To implement R-Type Processor

| Op | Opcode name | Value |
|---|---|---|
| 000000 | R-format | R-Type |
| 000010 | jmp | J-Tpye |
| 000100 | beq | I-Type |
| 100011 | lw | |
| 101011 | sw | |
| 010000 | addi | |




### III.1.2 CODE

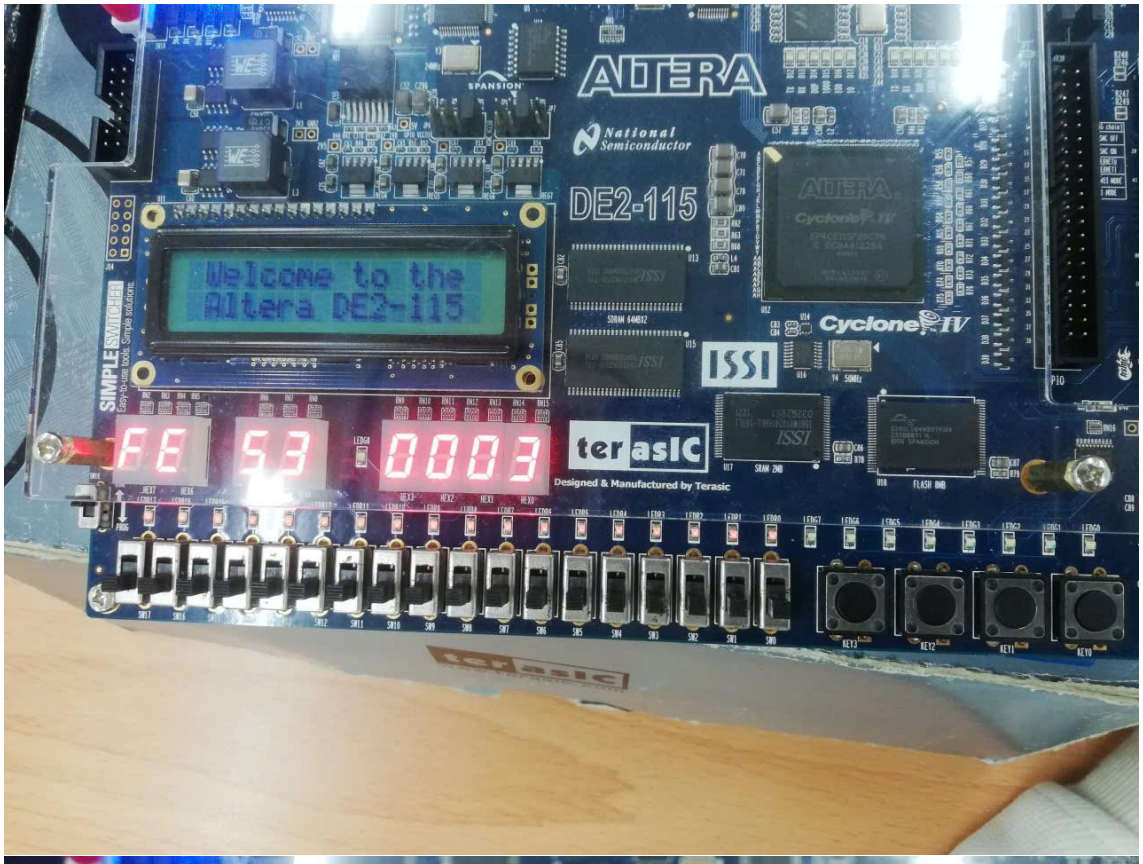
```
module R_Type_Proc(reset, clk,W_PC_out, instruction, W_RD1,
W_RD2,W_m1,W_m2,W_ALUout);
// Your code here


Endmodule
```

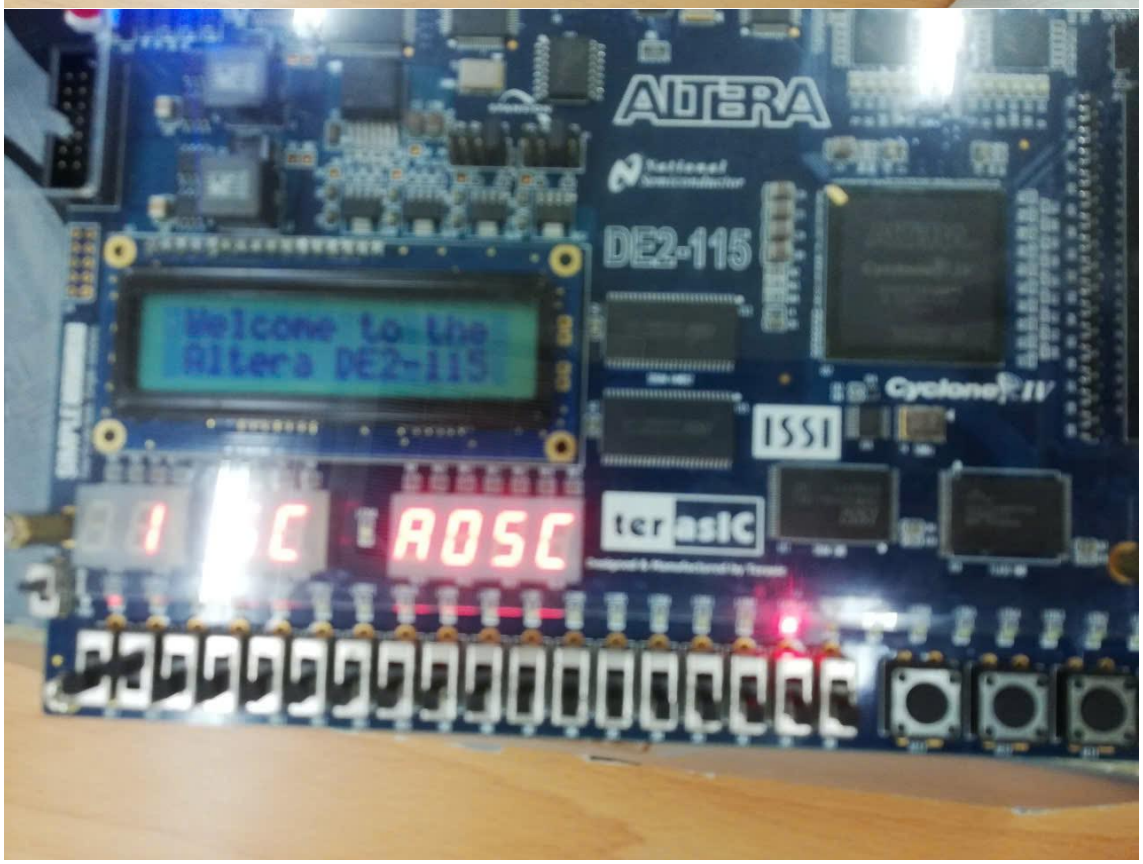
### III.1.3 LAB ASSIGNMENT

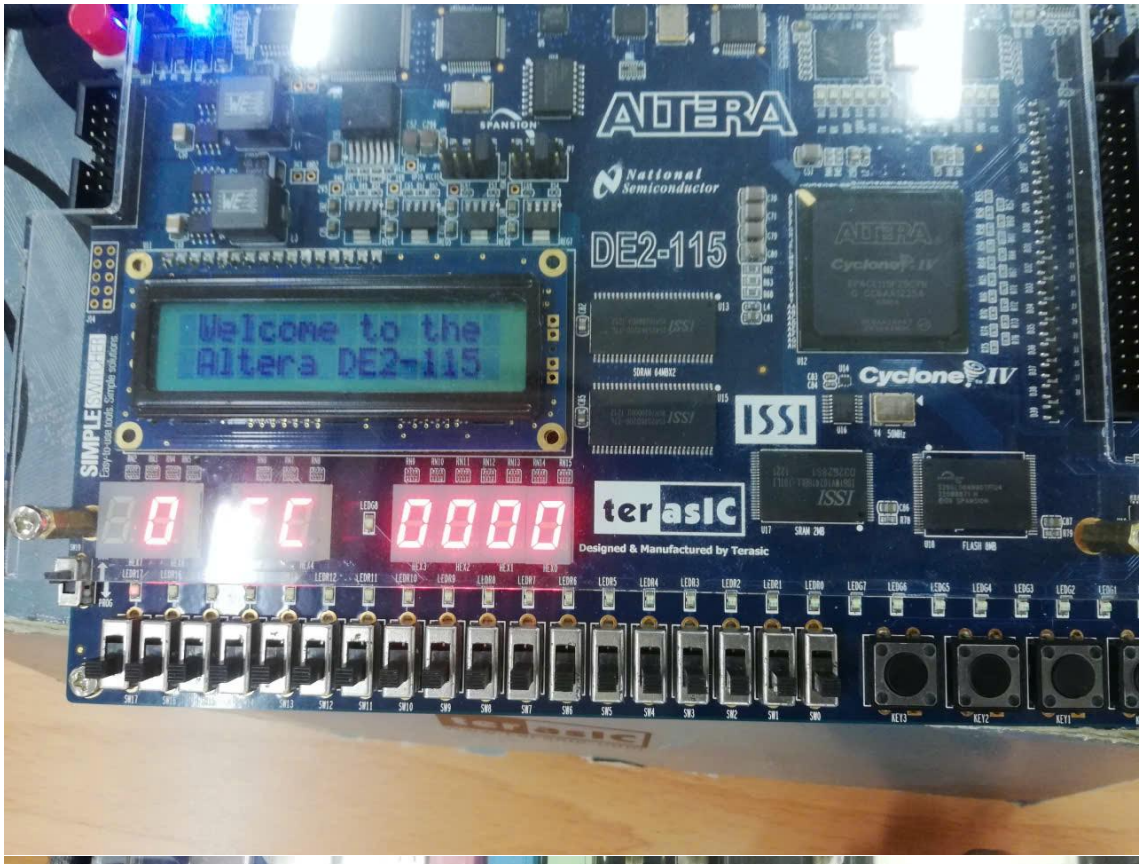
1) Write Verilog code to implement R_Type_Processor module
2) Write testbenches to verify R_Type_Processor, simulate and check the simulation output data.
3) Write the assembly code with Addittion, Substraction instructions, compile into binary macchinecode and test the operation of the designed R_Type_Processor processor

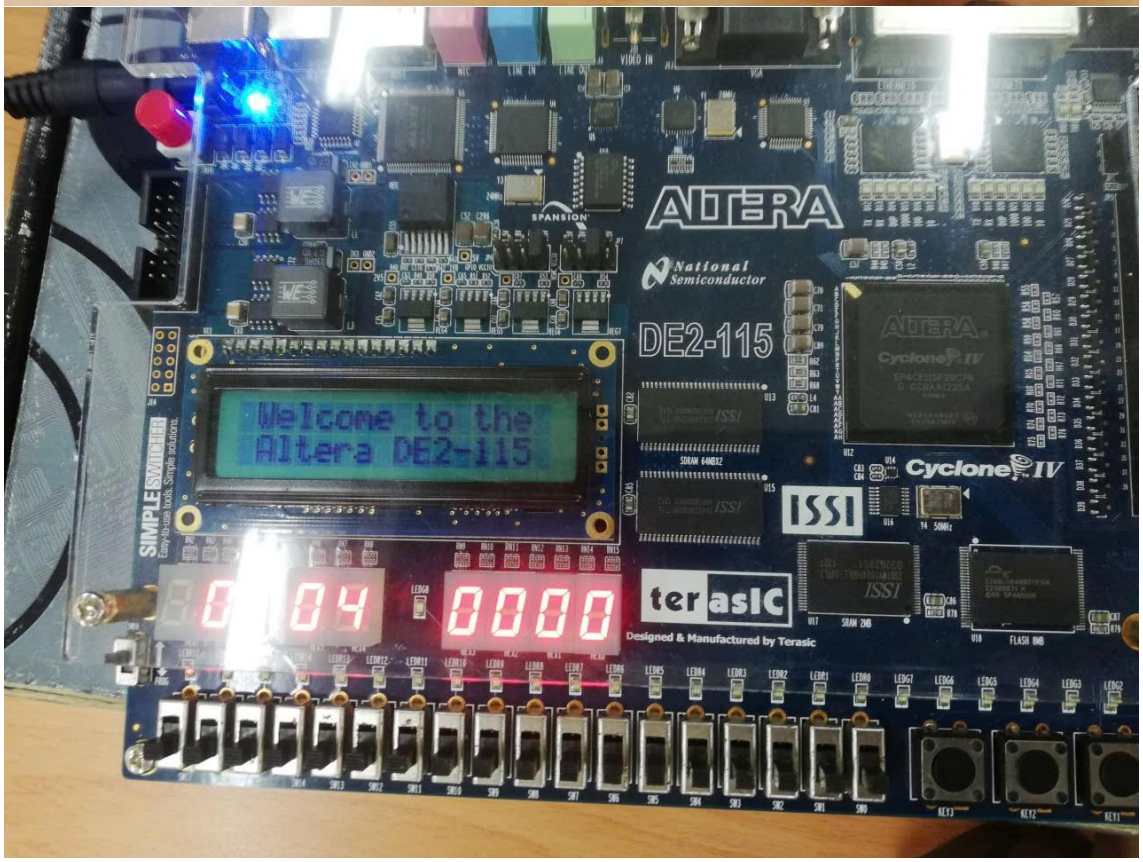
```
module lab6_ex1(
    input        CLOCK_50,    // PIN_Y2
    input  [3:0]  KEY,        // KEY[0]: Reset, KEY[1]: Step
    input  [17:0] SW,         // SW[1:0]: Chọn dữ liệu hiển thị
    output [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 // 8
LED 7 đoạn
);

    wire clk_cpu;
    wire reset = !KEY[0]; // KEY0 là Reset (Tích cực thấp)

    // --- BỘ CHỐNG DỘI PHÍM CHO KEY1 (Step) ---
    reg btn_reg1, btn_reg2;
    reg [19:0] debounce_cnt;
    always @(posedge CLOCK_50) begin
        if (debounce_cnt < 20'd1000000)
            debounce_cnt <= debounce_cnt + 1;
        else begin
            debounce_cnt <= 0;
            btn_reg1 <= KEY[1];
            btn_reg2 <= btn_reg1;
        end
    end
    assign clk_cpu = btn_reg2; // Xung nhịp CPU kích bằng nút nhấn

    // --- KẾT NỐI BỘ VI XỬ LÝ (DUT) ---
    wire [31:0] W_PC_out, instruction, W_ALUout, W_MemtoReg;
    wire [31:0] d1, d2, d3, d4;
    wire Jump;

    single_c_Proc CPU (
        .reset(reset),
        .clk(clk_cpu),
        .W_PC_out(W_PC_out), .instruction(instruction),
        .W_RD1(d1), .W_RD2(d2), .W_m1(d3), .W_m2(d4),
        .W_ALUout(W_ALUout), .W_MemtoReg(W_MemtoReg), .Jump(Jump)
    );

    // --- CHỌN DỮ LIỆU 32-BIT ĐỂ HIỂN THỊ ---
    reg [31:0] display_data;
```

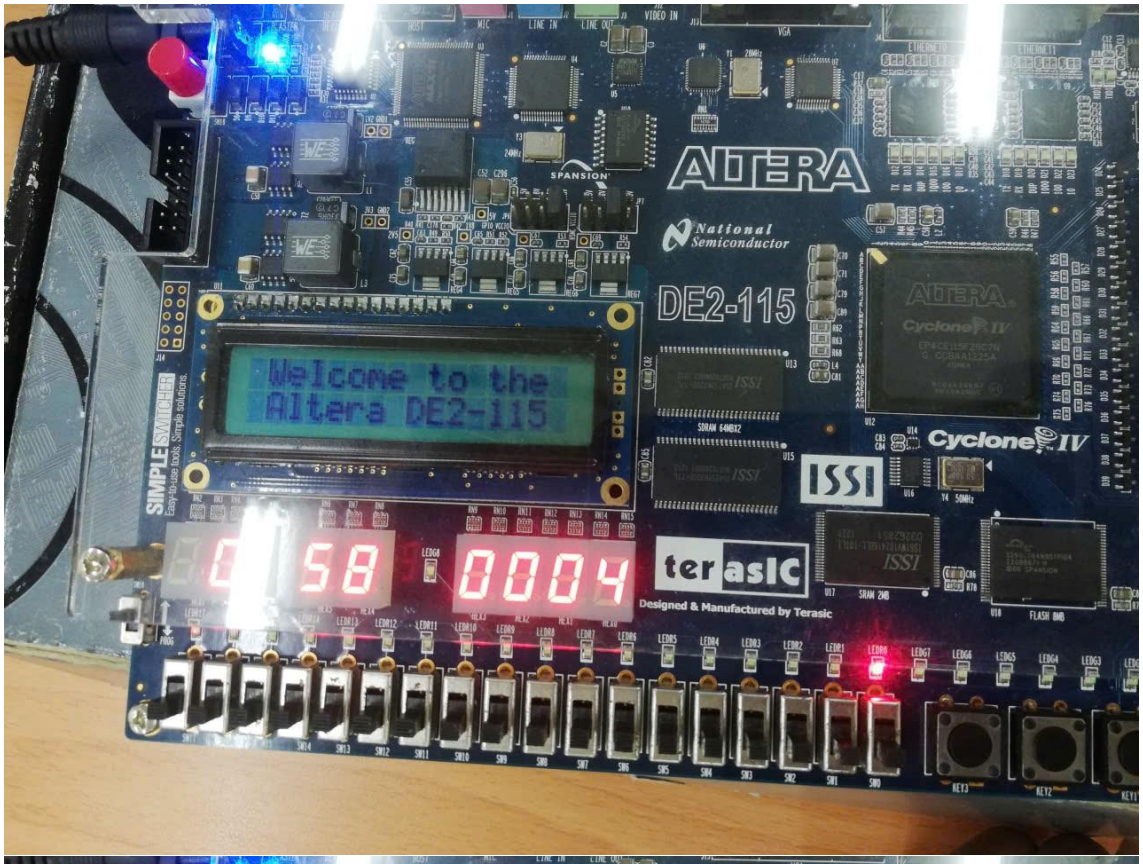
```
    always @(*) begin
        case(SW[1:0])
            2'b00: display_data = W_PC_out;     // Hiển thị PC 32-bit
            2'b01: display_data = W_ALUout;     // Hiển thị kết quả ALU 32-bit
            2'b10: display_data = W_MemtoReg;   // Hiển thị dữ liệu ghi vào Register
            default: display_data = instruction; // Hiển thị mã máy lệnh hiện tại
        endcase
    end

    // --- GỌI 8 MODULE HEX7SEG ĐỂ HIỂN THỊ TRỌN VẸN 32-BIT ---
    // Hiển thị từ HEX0 (4-bit thấp nhất) đến HEX7 (4-bit cao nhất)
    HEX7SEG h0 (.in(display_data[3:0]),   .out(HEX0));
    HEX7SEG h1 (.in(display_data[7:4]),   .out(HEX1));
    HEX7SEG h2 (.in(display_data[11:8]),  .out(HEX2));
    HEX7SEG h3 (.in(display_data[15:12]), .out(HEX3));
    HEX7SEG h4 (.in(display_data[19:16]), .out(HEX4));
    HEX7SEG h5 (.in(display_data[23:20]), .out(HEX5));
    HEX7SEG h6 (.in(display_data[27:24]), .out(HEX6));
    HEX7SEG h7 (.in(display_data[31:28]), .out(HEX7));

endmodule
```

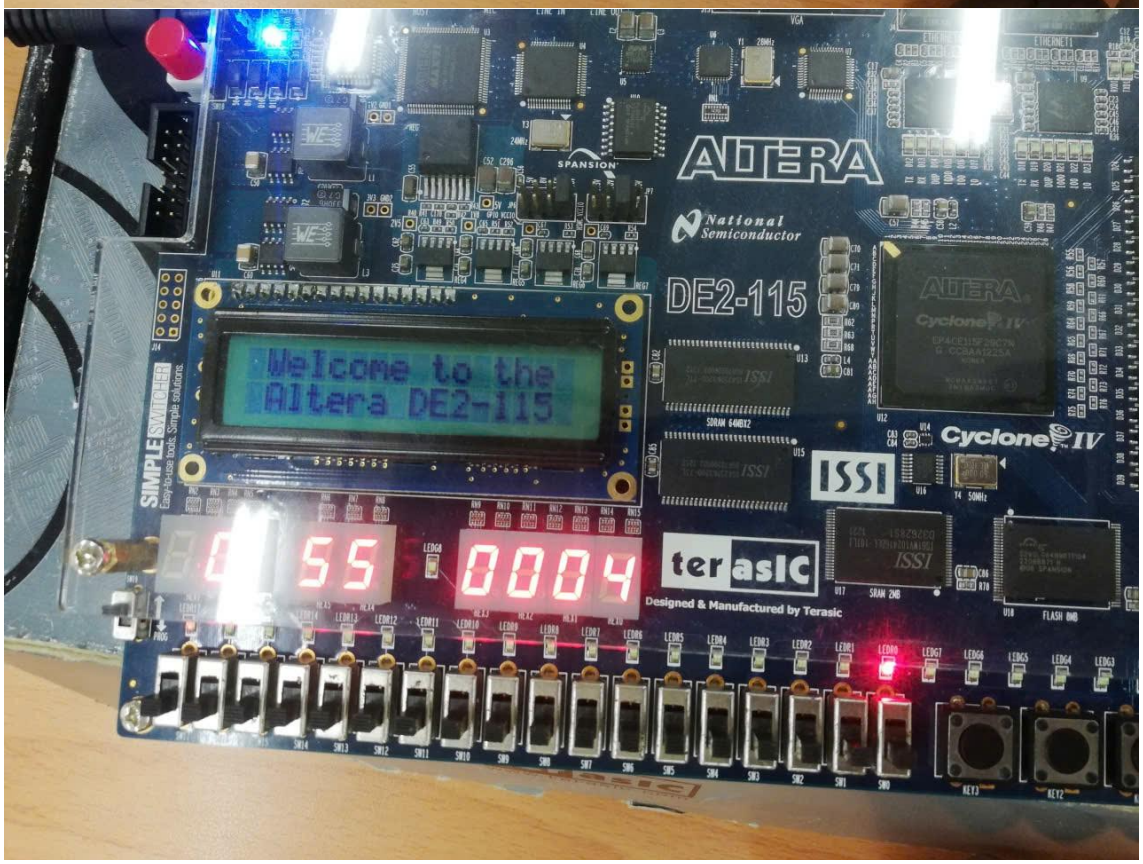

ALTERA
National Semiconductor
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
Designed & Manufactured by Terasic
FE 53 0003

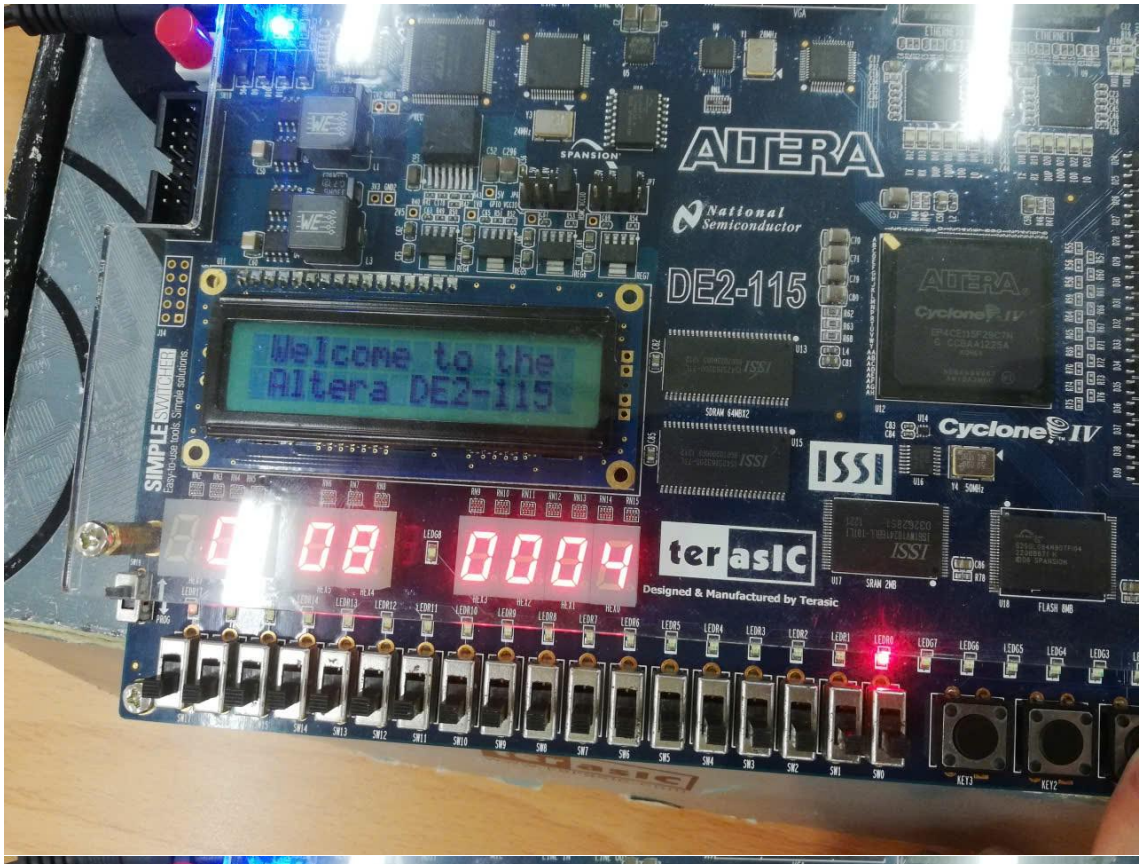

ALTERA
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
AOSC

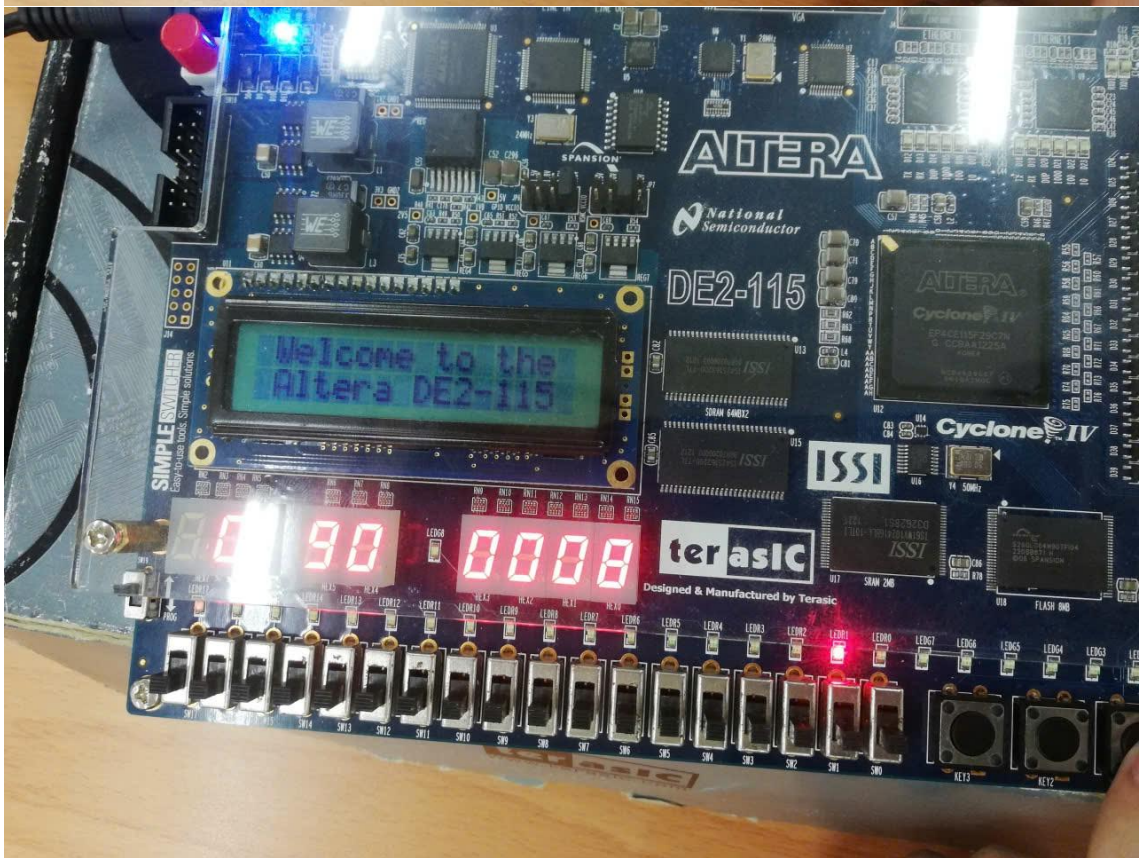

ALTERA
National Semiconductor
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
Designed & Manufactured by Terasic

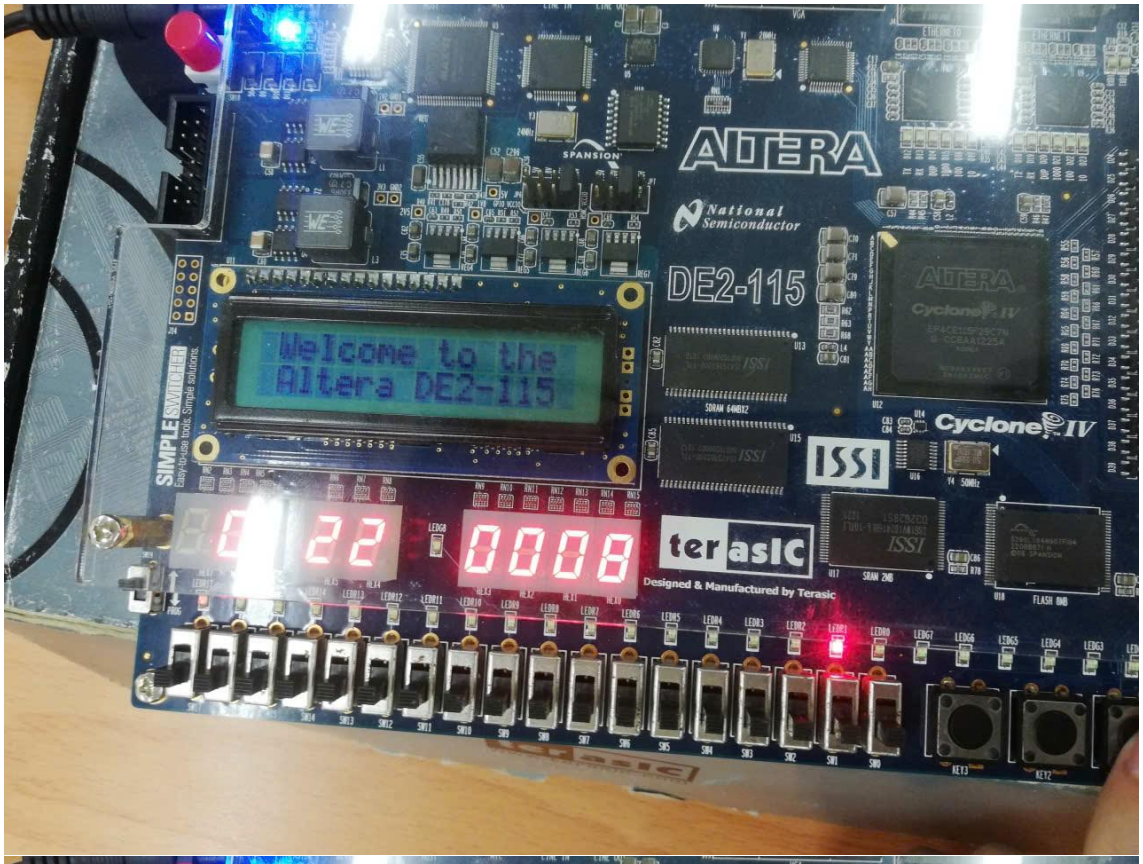

ALTERA
National Semiconductor
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
Designed & Manufactured by Terasic

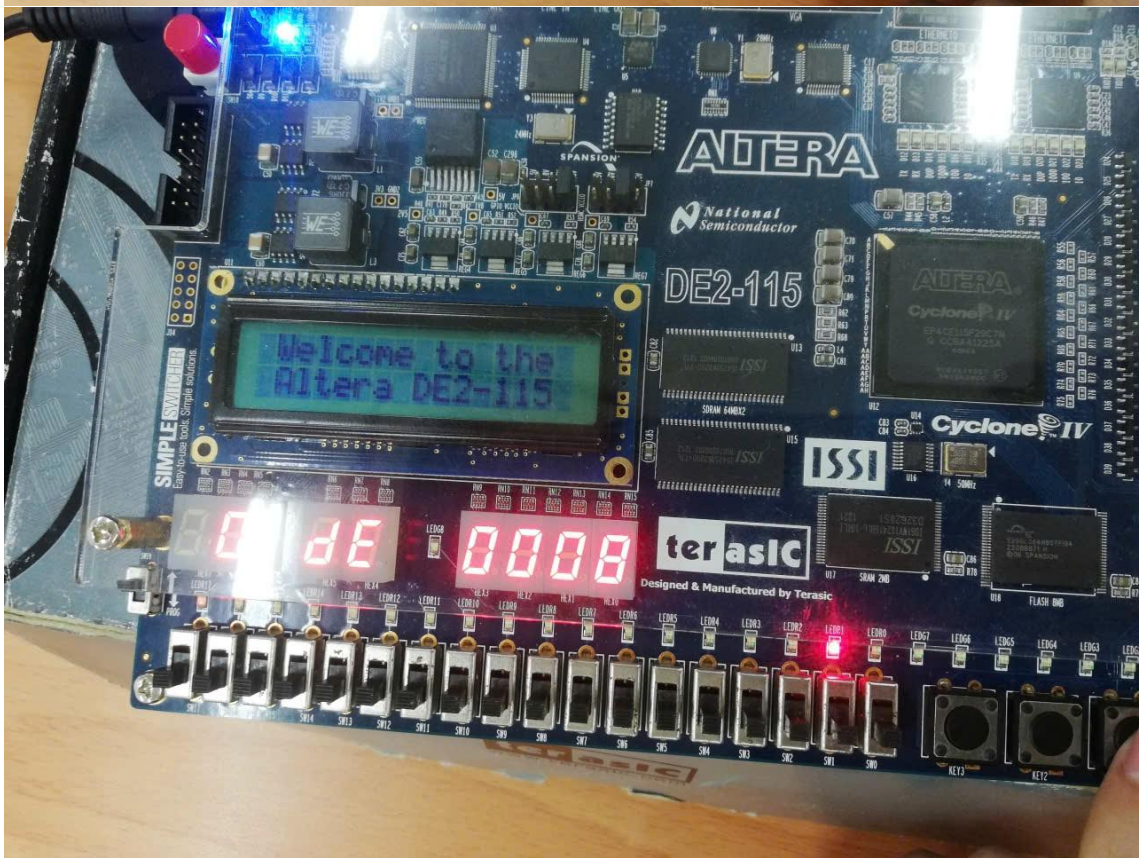

ALTERA
National Semiconductor
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
Designed & Manufactured by Terasic
SIMPLE SWITCH
Easy-to-use tools. Simple solutions.


ALTERA
National Semiconductor
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
Designed & Manufactured by Terasic
SIMPLE SWITCH
Easy-to-use tools. Simple solutions.


ALTERA
National Semiconductor
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
Designed & Manufactured by Terasic
SIMPLE SWITCHER


ALTERA
National Semiconductor
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
Designed & Manufactured by Terasic
SIMPLE SWITCHER


ALTERA
National Semiconductor
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
Designed & Manufactured by Terasic
SIMPLE SWITCHER


ALTERA
National Semiconductor
DE2-115
Welcome to the
Altera DE2-115
Cyclone IV
ISSI
terasIC
Designed & Manufactured by Terasic
SIMPLE SWITCHER